Hybrid Re-engineering

Linda H. Rosenberg, Ph.D.
Unisys Federal Systems/GSFC
Bld 6 Code 300.1
Greenbelt, MD 20771 USA
+1 301 286 0087
Linda.Rosenberg@gsfc.nasa.gov

Lawrence E. Hyatt
NASA Goddard Space Flight Center
Bld 6 Code 302
Greenbelt, MD 20771 USA
+1 301 286 7475
Larry.Hyatt@gsfc.nasa.gov

ABSTRACT
Traditional re-engineering applies reverse engineering to existing system code to extract design and requirements. Forward engineering is then used to develop the replacement system. Due to limited resources, many organizations, NASA included, are looking at the use of COTS software packages as a means of decreasing development time and costs. This paper briefly describes traditional re-engineering then discusses the emerging form of Hybrid Re-engineering. Hybrid re-engineering uses a combination of translation of existing code, COTS, and custom code to produce the replacement system. This paper discusses the advantages, disadvantages, potential risks and metrics for each component in Hybrid Re-engineering.

Keywords
Re-engineering, reverse engineering, forward engineering, risks, COTS, metrics

INTRODUCTION
Re-engineering is the examination, analysis and alteration of an existing software system to reconstitute it in a new form, and the subsequent implementation of the new form.[7] The process typically encompasses a combination of reverse engineering, re-documentation, restructuring, translation, and forward engineering. The goal is to understand the existing software system components (specification, design, implementation) to improve the subject system's functionality, performance or implementation. The primary risk is that the new target system will not have the same functionality as the existing system and will be lower in quality.[2]

Although objectives of a specific software re-engineering task are determined by the goals of the owners and users of a system, there are two general re-engineering objectives [10]:

1. Improve quality - Typically, the existing software system is of low quality, due to many modifications. User and system documentation is often out of date or no longer in existence. Re-engineering is intended to improve software quality and to produce current documentation. Improved quality is needed be to increase reliability, to improve maintainability, to reduce the cost of maintenance, and to prepare for functional enhancement. Object oriented technology may be applied as a means for improving maintainability and reducing costs.
2. Migration - Old working software may still meet the needs of the users, but be based on hardware platforms, operating systems, or languages that have become obsolete and thus may need to be re-engineered, transporting the software to a newer platform or language. Migration may involve extensive redesign if the new supporting platforms and operating systems are very different from the original, such as the move from a main frame to a network based computing environment.

This paper first looks at the generally accepted method for re-engineering, then describes an emerging form of re-engineering which the Software Assurance Technology Center (SATC) at NASA Goddard Space Flight Center (GSFC) has coined "Hybrid Re-engineering". After defining three "tracks" used in Hybrid Re-engineering, this paper discusses each track in-depth, looking at potential risks and how software metrics can be used to identify and mitigate those risks during the re-engineering process.

GENERAL MODEL FOR SOFTWARE RE-ENGINEERING
Re-engineering starts with the source code of an existing legacy system and concludes with the testing and implementation of the target system source code. Figure 1 diagrams a general model for software re-engineering that indicates the processes for all levels of re-engineering based on the levels of abstraction used in software development.[3] This is the standard format for re-engineering.

Re-engineering starts with the code and comprehensively reverse engineers by increasing the level of abstraction as far as needed toward the conceptual level, rethinking and re-evaluating the engineering and requirements of the

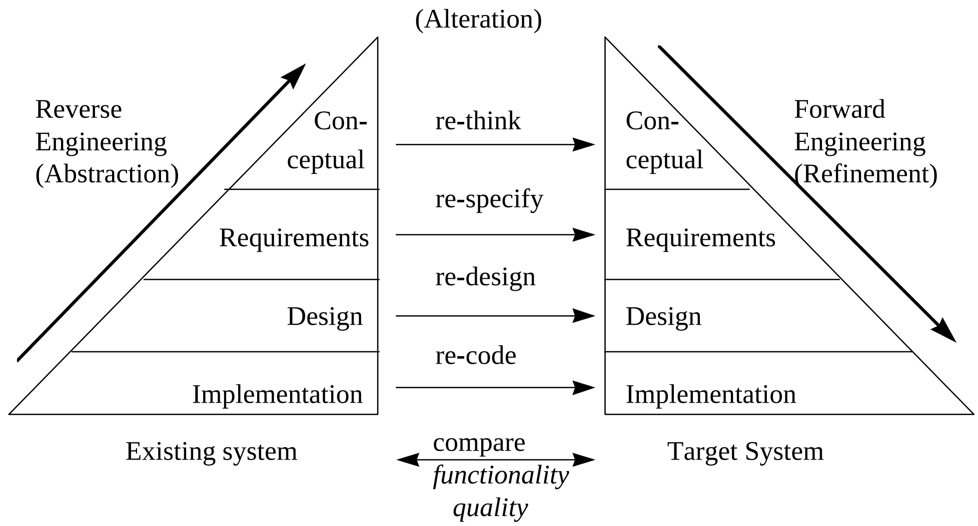

Figure 1: General Model for Software Re-engineering

current code, then forward engineers using a waterfall software development life-cycle to the target system. In re-engineering, industry reports indicate approximately 60% of the time is spent in the reverse engineering process, while 40% of the time is spent in forward engineering.[9] Upon completion of the target system, most projects must justify their effort, showing that the necessary functionality has been maintained while the quality has improved (implying improved reliability and decreased maintenance costs).

Re-engineering projects, like all projects, have limited funds and time and thus need to minimize the re-engineering cost while still maintaining system functionality and improving software quality. Many organizations, NASA included, are investigating the use of reusable software packages as a means of decreasing development time and costs and reducing development risks. Since the majority of the reusable software packages that are used are COTS (Commercial Off The Shelf), the term COTS will be used in this paper to represent this category of software.

In trying to decrease the cost of re-engineering, managers are considering the industry "rule of thumb" that 20% of the software causes 80% of the errors [4], and that approximately 15% of the system contains 85% of the functionality.[2] They are questioning the need to apply all the re-engineering phases to the entire system, as opposed to just the portion of the code causing most of the problems or containing most of the critical functionality. This thought process combined with time and budget constraints are leading to a new form of re-engineering, one that applies different levels of abstraction in reverse engineering (and different methods of alteration), to selected sections of the existing code.

HYBRID RE-ENGINEERING

The Software Assurance Technology Center (SATC) is working with NASA re-engineering projects to identify and minimize software development risks while improving the quality of the products. These projects are applying a combination of abstraction levels based on the needs of the project and its budget and schedule. The SATC has coined the phrase "Hybrid Re-engineering" to mean a combination of abstraction levels and alteration methods to transition an existing system to a target system.

Existing legacy systems are being re-engineered using the approach shown in Figure 2, an adaptation of the General Model for Software Re-engineering shown in Figure 1.

In Figure 2, three development tracks are utilized. The first track is a translation from existing code to a new language, operating system or hardware platform with no abstraction. The second track uses the existing code to identify requirements that can be satisfied by the application of COTS packages. The third track is the development of custom code for project unique requirements that cannot be satisfied by either other track, and to "glue" together the other components.

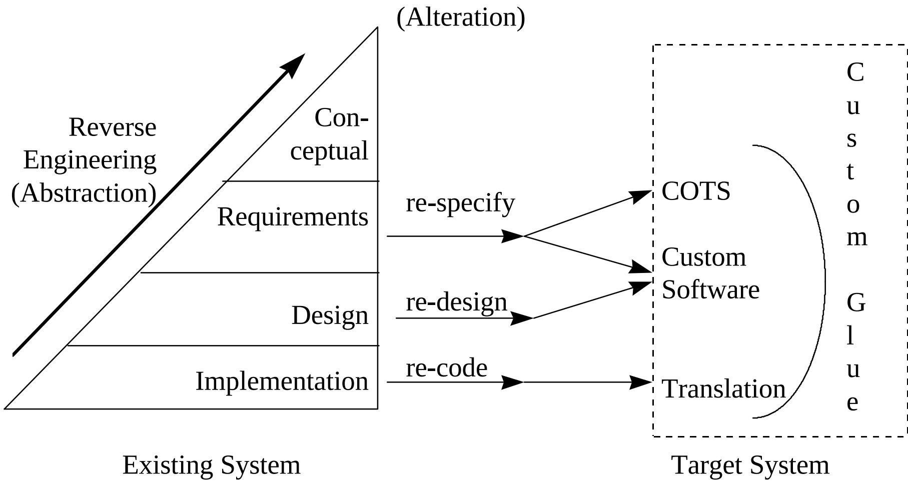

Figure 2: Hybrid Re-engineering Tracks

Re-engineering as a development methodology has inherent risks such as schedule, functionality, cost, and quality. Hybrid re-engineering was developed to decrease some of these risks since COTS packages are expected to have high reliability and require minimal development time. Another method of decreasing time and cost through Hybrid re-engineering while maintaining functionality is through a straight translation of part of the current code to the new language or operating system.

Hybrid re-engineering is innovative, combining three distinct re-engineering efforts, hence the risks generally associated with re-engineering can increase by combining the risks inherent to each track. Since hybrid re-engineering is combining products from different development tracks (COTS, custom software and translated software), one new risk is the interface and interoperability of the products. For example, data transfer between products can cause compatibility and timing problems; COTS packages may not work exactly as anticipated.

In general metrics can be used by management to improve software development efforts and minimize the risks. Metrics can indicate how well a project is meeting its goals.[5] In hybrid re-engineering, metrics can support the justification for decisions on track selection for different software functions and components.

The following sections describe each of the three hybrid re-engineering tracks. After each track is described, the risks associated with the track are identified and appropriate metrics defined.

Translation Track Hybrid Re-engineering

Figure 3 is a diagram of a "typical" software system that has been in use for some period of time. In this re-engineering example, we assume the project is moving from FORTRAN to C++ (but not necessarily to an object-oriented design).

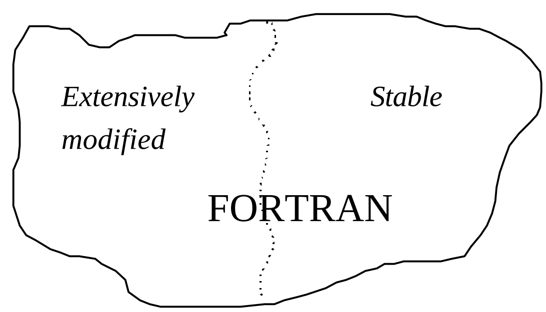

Figure 3: Current Software System

In looking at the current software system, the manager sees two classifications of code, some stable code that has had minimal modifications and whose requirements have remained stable, and some code that has under gone multiple changes and has become unstable, unreliable, and costly to maintain. Re-engineering the stable code may not require total reverse engineering; it might be feasible to simply re-code this portion into the new language or new environment. This process constitutes the translation track of hybrid re-engineering.

In this track, shown in Figure 4, the code in the existing system that is relatively stable, having had minimal changes to the original design and architecture, must be identified. This can be accomplished by an analysis of the code and of change reports.

Many source code translators are available to support the transport of code from one language or operating system to

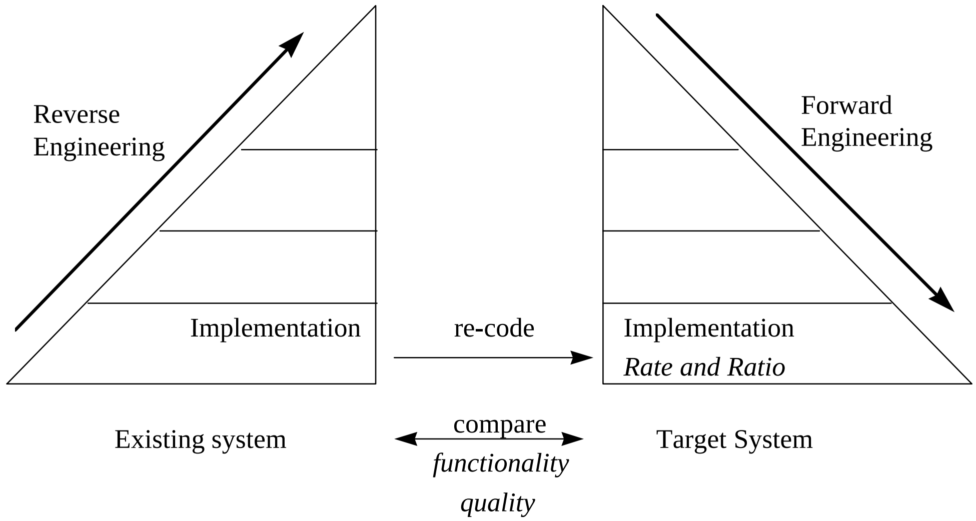

Figure 4: Translation Track Hybrid Re-engineering

another.[12] An advantage of simply translating code is that maintenance personnel will still understand the program flow since it remains unchanged. This will aid them in becoming effective in the new language. Source code translators may not be adequate alone since line-by-line translators don't take advantage of target language semantics, constructs, etc., often resulting in code know as "C-TRAN" - C syntax on FORTRAN structures.[10]

In the Translation track, the primary risk is the quality of the resulting code. When transitioning from one language to another, the code can have the syntax of the new language but none of the structures or new features. While the initial legacy code was of adequate quality, this does not guarantee that the resulting code will have the same quality. The results of translation must be reviewed for quality. If the quality is not adequate, code may have to be improved. If 20 - 30% of the translated code must be changed to improve its quality or to meet standards, the code should not be used and those functions or components should be re-engineered using another of the three tracks.[11]

Prior to re-engineering, in identifying candidates for translation, metrics such as the logical and calling complexities provide valuable information on structure and coupling and suggest candidates for translation. In identifying components that have been extensively maintained, change or problem reports supply the data. Tracking the criticality of functions will assist in making tradeoff decisions. Quantifying the functionally of the legacy system will provide a basis for estimating how complete the new system is during development and provide estimations as to when the target system will be complete.

One method being tested by the SATC and other companies to track progress in re-engineering uses function points as a measure of functionality. In this application, using the basic function point counting method described by Albrecht, an approximate estimate can be gleaned as to the type and count of tasks[1]. This can be used as a starting point in comparing progress in transferring functionality between the original system and the target system. Tools are available to count function points from COBOL code and the SATC is working to develop a tool for FORTRAN and C. In evaluating the progress of the translation, one measure might be the rate at which functionality, approximated through function points, is moving from the existing system to the target system. The ratio of code translated can also serve as a measure of progress.

Once the re-engineering is complete, it is important to verify that the functionality is retained in the new system, as well as the quality of the code has improved, hence implying improved maintainability. There is no simple method to ensure functionality has transitioned between systems. The most commonly used method involves running test cases on the original system and then repeating the tests on the completed target system.

COTS Track Hybrid Re-engineering

In the COTS track of Hybrid re-engineering, shown in Figure 5, requirements and functions must be identified that can feasibly be implemented using COTS.

After applying the techniques of Reverse Re-engineering to identify the requirements, it is important to separate those requirements that must be contained in the target system from those requirements that users want in the new system because they have become habits or are comfortable

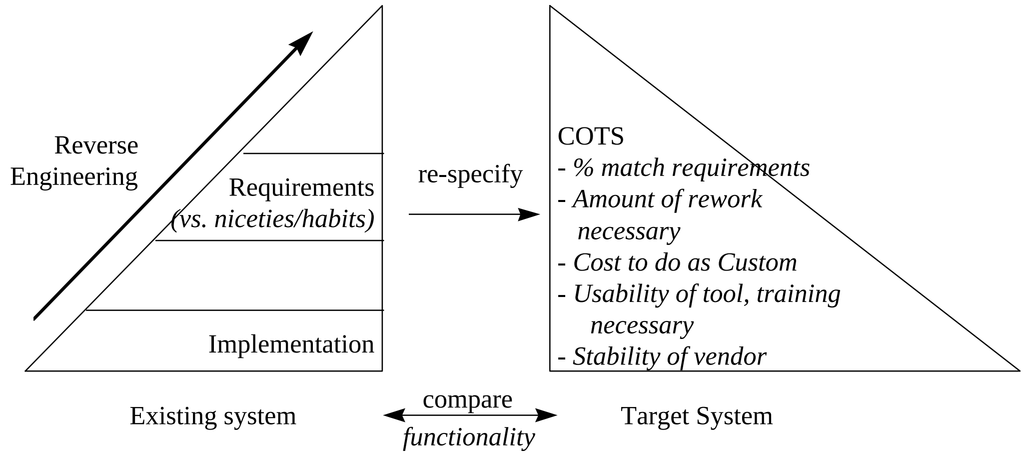

Figure 5: COTS Track Hybrid Re-engineering

with those features. This separation of requirements into "necessary" and "nice" is critical to COTS selection.

The advantage to using COTS is the decreased development time and increased reliability. Testing is still needed during integration, but less than for code developed from scratch. The disadvantage can be the number of requirements satisfied by the package. For example, if in the existing system a specific field has 10 characters and the COTS package only allows 8, is this acceptable? Was the 10 character field arbitrary or does it represent a "hidden" business requirement that should not be altered? In addition, users are often resistant to change, even ones that have little impact, such as different icons or keystrokes to call a function. This must be taken into account for the package to have comparable usability and functionality.

Although the use of COTS software decreases the development time and increases the reliability, COTS also introduce additional risks. A major risk is that the package will not perform as anticipated or advertised, that it is unreliable, immature or incomplete. The package may also undergo frequent manufacturer version enhancements requiring constant upgrading. In the worse case, changes may alter or remove functions needed for the system. The COTS may require modifications or supplementation to match the requirements, causing increases in schedule and decreasing reliability. In addition, the use of a COTS may limit further enhancements to the system, since changes in the COTS provided functions may not be possible due to legal issues. The stability of the vendor should also be part of the evaluation process since it may be necessary for them to make later required changes. An additional cost may be incurred due to the unfamiliarity with the COTS. Simple changes to usability, such as new icons, will require additional training time.

Some metrics are applicable to assist in decreasing the risks associated with the selection of COTS packages,. It is important to first identify what percentage of the desired requirements the package totally meets, and which requirements the package partially meets. This information can then be used to determine how much rework or supplementation the package requires in order to totally fulfill the system requirements. Modification or supplementation will impact the schedule and budget, and may well impact maintainability and reliability. After all of these evaluations are complete, the cost to develop from scratch should also be estimated, including testing time, and compared to the total costs of the COTS.

In this track, once the COTS have been implemented, the functionality of the existing system must be compared to the functionality of the target system. The process of comparison must be based on testing, as was done for translated code functions.

Custom Track Hybrid Re-engineering

The Custom track of Hybrid Re-engineering, shown in Figure 6, is similar to traditional re-engineering since new code is derived from the existing legacy system.

In this track, reverse engineering is first performed. Those functions that are not satisfied by COTS packages or through translated code must be identified, and its requirements and design extracted. Forward engineering is then performed. This begins with requirements analysis, with the objective of identifying requirements that are not needed. The process is then similar to any development process, beginning with developing a new design, with object-oriented structure if desired, then implementing the code and doing comprehensive testing.

The advantages to the custom track is that the resulting code should meet its requirements exactly. The developed

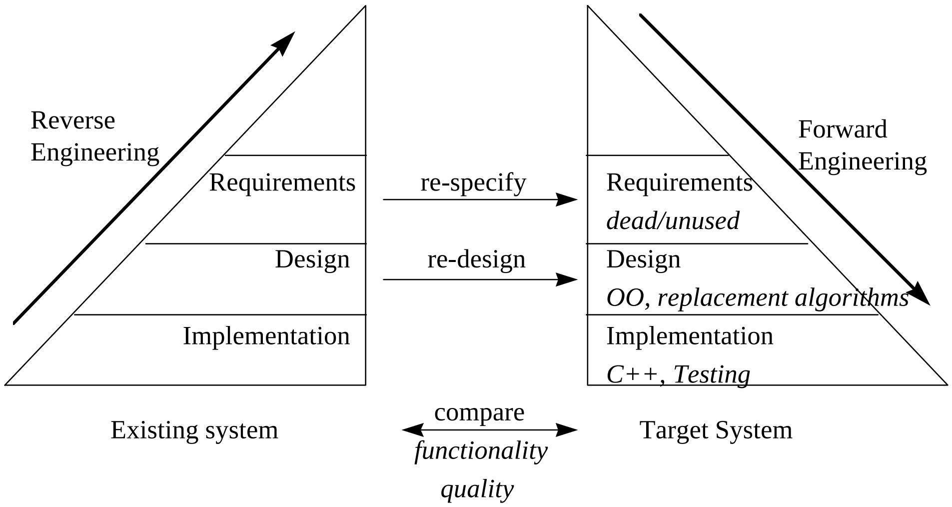

Figure 6: Custom Track Hybrid Re-engineering

code should be of high quality and well structured, requiring little maintenance. The disadvantages are similar to standard software development, in that the code might not be reliable requiring additional testing, and that the development/testing process may exceed time and cost budgets.

Custom code has the same inherent risks that any code has, to quality, reliability, and schedule. Since most of the functions of the legacy system identified as unique to the system will be done with custom code, the risk is that one of the unique features will not be identified and hence the functionality of the new system will be incomplete. Features identified as critical to the system that are accomplished with custom code will require extensive testing.

Metrics for this track are a combination of both process and product metrics. Prior to reverse engineering, the quality of the existing system should be evaluated for later comparison to the target system (as discussed previously). Effort expended should be tracked to assist in the evaluation of the cost to re-engineer. This can also be used to approximate schedule completion using the estimate that 60% of the time is in reverse engineering. Once requirements are re-specified, their quality can be evaluated to determine testability.[13] Also discussed previously was the use of function points as a means of calculating the rate of functionality transfer to the target system. Code analysis tools can be used to evaluate the quality of the code as it is being developed and identify the risks.[5] In testing, discrepancy or error rates help in evaluating reliability.

In this track, both the functionality and the quality are compared between the existing system and the target system.

HYBRID RE-ENGINEERING APPROACH

Hybrid re-engineering requires an approach similar to traditional re-engineering, but with additional considerations. When starting to re-engineer, initial justifications for re-engineering such as costs and quality are developed and expectations, such as return on investment, are stated. An analysis of the legacy system should be done to determine the feasibility of hybrid re-engineering. The analysis of the legacy system should provide a guideline in identifying optimal strategies (translation, COTS, etc.), and projecting the cost of the target system. Once the decision on using a hybrid re-engineering approach is made, additional analysis is needed.

The first step in a hybrid re-engineering approach is to investigate the requirements and constraints of the development. These factors include setting a time table for reverse and forward engineering. Time must be built in to investigate available COTS, including hands on testing of the COTS. While forward engineering development time should decrease with the use of COTS and the translation of code, additional time will be needed for testing the integration and interface of the products of the three tracks. Budget constraints must also be considered; how much can be spent on COTS that provide required features versus those that provide desired features. Management mandated and organization needs must also be identified. As the three tracks are developed, tradeoffs will be necessary so it is important to prioritize requirements.

The next step is to do an in-depth analysis of the legacy system, focusing on functionalities and code segments suitable for each of the three tracks (Translation, Custom, COTS). In generic re-engineering, an analysis of the existing system is usually done to provide an evaluation of the quality of the existing system and maintenance costs. This information is used to justify the costs and improvements at the conclusion of the re-engineering effort. While these reasons are still relevant in hybrid re-engineering, additional features of the legacy system must now be investigated. During the assessment of the legacy system, sections and functionalities must be identified. These must be further assessed to determine what documentation is available to identify the required features versus what is no longer needed or what users have become accustomed to. Code sections must be ordered by the cost of maintenance, and the quality of the current structure. Functions that are unique to this project must be identified. All of these components will be used to identify which hybrid re-engineering track will be applied to the code section.

Once the code has been divided into the development tracks, each track will proceed independently as discussed. The schedule for track completion will differ based on tasks. As the tracks conclude various tasks, the merging of the final products can begin. For example, the custom glue can be started once the COTS have been selected. Training on these packages can also begin.

Once the system is complete and all tracks merged, two tasks remain: testing and justification. First, comprehensive System and Integration testing must be performed to ensure all components work together as a cohesive unit and to ensure all functionality of the existing system was transferred to the new system. Second, justification for the re-engineering is usually required - do the benefits gained justify the cost. Some anticipated benefits, such as improved maintenance and operational costs, can only be demonstrated indirectly through the improved quality. Improved quality can initially be demonstrated by a metric analysis of the legacy system compared to the new system. As the new system is put into operation, additional metrics can be used to verify the improvements.

HYBRID RE-ENGINEERING RISKS

All software development has inherent risks to schedule and cost. Hybrid re-engineering, as a software development methodology, is also susceptible to them. Hybrid re-engineering, because of its composition of the three diverse development tracks, is subject to all of the risks that were discussed within each track description. Also, Hybrid re-engineering as a unique software re-engineering methodology has additional risks to functionality and quality; the functionality of the existing system must be preserved in the new system, and the quality must improve, implying a decrease in operational and maintenance costs. With all of the risks in Hybrid re-engineering, why bother, why not just treat it as a new software development effort and omit the re-engineering all together? Because of the benefits.

HYBRID RE-ENGINEERING BENEFITS

In general, re-engineering is performed as opposed to building a new system because of the invisible business application procedures and logic that are built into the software. These processes might be deeply embedded in business procedures as simple as a field length or as complicated as a mathematical algorithm; the only source of this information is in the legacy code. A second justification for re-engineering versus building is the development and maintenance costs of the legacy system; the time spent developing logic and components should not be wasted. In re-engineering, the existing system is re -implemented and instilled with good software development methodologies, properties, and new technology while maintaining existing functionality. Reliability and maintainability are also improved.

Hybrid re-engineering has the additional benefits of a reduced development schedule, hence reduced costs. The development schedule is shortened first by minimizing the amount of reverse engineering (recall, reverse engineering is 60% of the effort). The translation track uses minimal reverse engineering time since work is done in the lowest level (Figure 2). The use of COTS decreases the forward engineering development and test time and thus the costs. The use of properly selected COTS also increases the reliability since these packages have been extensively tested.

HYBRID RE-ENGINEERING METRICS

Metrics, when properly applied, provide managers information about their project to help mitigate risks.[5] It is logical therefore, to discuss some of the re-engineering phases where metrics provide valuable information. Previously we have identified metrics applicable for each track in hybrid re-engineering. In this section, we will discuss metrics applicable to the entire project, not just one track. Metrics provide information on the quality, functionality, and on track selection, a prime areas of risk.

At the start of the re-engineering effort, the legacy system must be quantified. There are two objectives: identify the amount of functionality and the quality of the existing system. By quantifying the functionality, scheduling estimates are more accurate; during development, completion can be estimated by the percentage of functionality transferred to the new system. Functionality is also important at the conclusion of the project, measuring how much functionality is contained within the new system. The SATC and others are working with function points as a means of estimating functionality. Function points are comparable across languages, and time estimates based on function points are available.[6] For COTS packages, functionality might be measured by the number of requirements satisfied.

Quality is harder to measure and few software developers agree totally on the appropriate metrics. The SATC has a group of metrics it applies to projects to evaluate the quality. These metrics evaluate the project at the module level (procedure, function, class or method). The size is measured by counting the number of executable statements. The readability is measured by the comment percentage. The complexity is measured by the cyclomatic complexity (McCabe). The coupling is measured by the calling complexity (fan in / fan out).[8] Although there are many other metrics available, these have been successfully applied to many projects at GSFC NASA. One final measure of quality is the reliability, the number of errors found, and the projected number of errors remaining. These metrics can be used for both the translation track and the custom code track. When the components are combined, the numbers of errors found and the projected number of errors remaining can be applied to the whole system.

SUMMARY

The number of large systems being built from scratch is diminishing, while the number of legacy systems in use is very high. Rapid changes in the computer industry continually introduces new hardware and software making older systems obsolescent and difficult to maintain. Businesses do not want to re-develop from scratch; too many business decisions are built into the legacy systems. Hence re-engineering. But project development is always short on time and money, making the need to look at alternatives necessary. The use of COTS packages is seen as a way to increase reliability while decreasing development and test time. Translation of code is a means of decreasing time and cost. This has resulted in a combination of the development methods into a form of hybrid re-engineering.

FUTURE WORK

The field of software re-engineering is new, and its risks and metrics are not well understood. The newest area, hybrid re-engineering, is even less well known. The SATC is working to define metrics to quantify the quality of legacy systems and of the re-engineered products, and to provide reliable measures of progress. It is in this last area, the measure of progress, that we are experimenting with function points.

REFERENCES

[1] Albrecht, A., Gaffney, J. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", *IEEE Transactions on Software Engineering*, 11/83.

[2] Arnold, R., *Software Reengineering*, IEEE Computer Society Press, 1993.

[3] Byrne, E., " A Conceptual Foundation for Software re-engineering", Conference on Software Maintenance, 1992.

[4] Grady, R., *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, 1992.

[5] Hyatt, L., Rosenberg, L., "A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality", 8th Annual Software Technology Conference, UT, 4/96.

[6] Jones, C., *Applied Software Measurement*, McGraw Hill, Inc., 1991.

[7] Manzella, Mutafelija, "Concept of re-engineering Life Cycle", IEEE, 1992.

[8] Rosenberg, L., Hyatt, L., "Developing a Successful Metrics Program", European Space Agency Symposium, 3/96.

[9] Ruhl, M.K., Gunn, M.T., "Software Re-engineering: A Case Study and Lessons Learned", NIST Special Publication, 9/91.

[10] Sneed, S., " Planning the re-engineering of Legacy systems", IEEE Software, 1/95.

[11] Software Technology Support Center, "STS Reengineering Technology Report, Vol 1", Hill Air Force Base, UT, 10/95.

[12] Software Technology Support Center, "STS Reengineering Technology Report, Vol 2", Hill Air Force Base, UT, 10/95.

[13] Wilson, W., Rosenberg, L., Hyatt, L., "Automated Analysis of Requirements Specification", Fourteenth Annual Pacific Northwest Software Quality Conference, Oregon, 10/96.